Gewächshaus

Aufgaben

1 Ein Gewächshaushersteller möchte eine Serie automatisierter Gewächshäuser herausbringen. Hierzu benötigt er eine Software, die die Sensoren im Gewächshaus überwacht und abhängig von den erfassten Werten und den von den Benutzerinnen bzw. Benutzern (im Folgenden Benutzer genannt) eingestellten Schwellwerten verschiedene Prozesse im Gewächshaus steuert.

1.1 Zusätzlich zu den in Material 1 bereits gegebenen Anwendungsfällen liegt folgende Anwendungsbeschreibung vor:
Die Gärtnerin bzw. der Gärtner (im Folgenden Gärtner genannt) kann das Bewässerungssystem (im Folgenden Bewässerung genannt) konfigurieren. Dazu wählt er die Wassermenge und einen Modus, nach dem bewässert werden soll, aus. Für die Modi „täglich" und „wöchentlich" muss eine Uhrzeit gewählt werden und für den Modus „wöchentlich" der Wochentag. Für den Modus „Bodenfeuchte" muss der Schwellwert für die Bodenfeuchte ausgewählt werden, bei dem die Bewässerung starten soll.
Neben dem Gärtner gibt es noch eine Hilfsgärtnerin bzw. einen Hilfsgärtner (im Folgenden Hilfsgärtner genannt). Sowohl Gärtner als auch Hilfsgärtner können die Lichtverhältnisse und die Temperaturen im Gewächshaus abfragen. Der Hilfsgärtner kann allerdings keine Konfiguration vornehmen. Um das System benutzen zu können, müssen sich alle Akteurinnen bzw. Akteure (im Folgenden Akteure genannt) mit Benutzername und Passwort einloggen.
Entwickeln Sie ein aussagekräftiges Anwendungsfalldiagramm (Use-Case-Diagramm) für diese Anwendungsbeschreibung. Bestimmen Sie dabei die menschlichen Akteure, die Beziehungen zwischen Akteuren und Anwendungsfällen und eventuell vorhandene Vererbungs-, include- oder extend-Beziehungen.

(12 BE)

1.2 Das Gesamtsystem besteht aus zwei Teilen. Der eine Teil, das Gewächshaus, ist bereits im UML-Klassendiagramm in Material 1 vorgegeben. Der zweite Teil, die Verwaltung der Benutzer, gestaltet sich wie folgt.
Die zentrale Klasse ist die `Benutzerverwaltung`. Es gibt zwei Arten von Benutzern, die Administratorin bzw. den Administrator (im Folgenden Administrator genannt) und den Gärtner. Beide besitzen einen Benutzernamen und ein Passwort. Der Hilfsgärtner wird über ein Attribut `typ` in der Klasse Benutzer unterschieden.
Ein Administrator kennt die `Benutzerverwaltung` und besitzt zusätzlich zu den Eigenschaften, die er als Benutzer hat, drei Methoden. Eine Methode legt einen neuen Benutzer mit Benutzername, Passwort und Typ an. Eine Methode dient dazu, das Passwort zu ändern, das zu einem Benutzer gehört, der durch seinen Benutzernamen identifiziert wird. Die dritte Methode ändert den Typ des Benutzers ebenfalls anhand des Benutzernamens.
Der Gärtner kennt die `Gewaechshaussteuerung` (Material 1) und besitzt Methoden zum Konfigurieren des Gewächshauses und zum Abfragen der Sensoren analog zu denen der `Gewaechshaussteuerung`, allerdings ohne dass er den Benutzer explizit übergeben muss.
Für den Hilfsgärtner aus Aufgabe 1.1 wird keine eigene Klasse angelegt, seine Berechtigungen werden über die Gruppenzuordnung geregelt.

Modellieren Sie zu der obenstehenden Beschreibung ein UML-Klassendiagramm mit geeigneten Datentypen, Multiplizitäten sowie den Getter- und Setter-Methoden, die in der Beschreibung explizit verlangt werden, im Design-Modell.

Hinweise: Konstruktoren sind nicht zu modellieren. Übernehmen Sie nur notwendige Klassen aus Material 1 ohne Methoden.

(10 BE)

1.3 Zur Konfiguration der Bewässerung im Modus „Bodenfeuchte“ dient die Methode `konfiguriereBewaesserung(Benutzer benutzer, int wassermenge, int schwellwert)`. Sie fragt zuerst bei der Benutzerverwaltung nach, ob der Benutzer berechtigt ist, die Bewässerung zu konfigurieren. Ist er berechtigt, werden die Wassermenge und der Schwellwert gesetzt. Damit ist die Konfiguration abgeschlossen.
Die Methode `holeFeuchtigkeit(Benutzer benutzer)` berechnet den Mittelwert der Messwerte aller Feuchtigkeitssensoren und gibt ihn zurück. Auch diese Methode prüft zuerst die Berechtigung des Benutzers. Besitzt der Benutzer keine Berechtigung, gibt sie als Ergebnis den Wert -1 zurück.
Implementieren Sie die Klasse `Gewaechshaussteuerung` mit allen Attributen, dem Konstruktor und den oben beschriebenen Methoden gemäß dem UML-Klassendiagramm in Material 2 so, dass die beschriebenen Vorgänge durchgeführt werden können.

Hinweise: Die übrigen Methoden sind nicht zu implementieren. Die Berechtigungen sind Aufgabe 1.1 zu entnehmen.

(15 BE)

1.4 Die `Gewächshaussteuerung` ruft in regelmäßigen Abständen auf allen Sensoren die Methode `initialisieren()` auf. Diese Methode initialisiert den jeweiligen Sensor für eine neue Messung, lässt den Sensor die Messung durchführen und speichert das Messergebnis in dem Attribut `wert` ab.
Im Folgenden ist der Ablauf anhand eines konkreten Temperatursensors genauer beschrieben.

- Ein Objekt der Klasse `Temperatursensor` baut als Client eine Netzwerkverbindung zu dem physikalischen Temperatursensor (`physTempS`) auf, der als Server fungiert.
- Der Server mit der Adresse 192.168.5.8 lauscht auf Port 4321 auf Verbindungen.
- Sobald eine Verbindung aufgebaut ist, erwartet er vom Client das Schlüsselwort „Temp“.
- Hat der Client das Schlüsselwort gesendet, so sendet der Server den Temperaturwert, den er in der Variable `temperatur` gespeichert hat, an den Client.
- Wurde ein falsches Schlüsselwort gesendet, so wird der Wert „-999“ übertragen.
- Der Client konvertiert den übertragenen Wert mittels einer eigenen Methode `convert(…)` in das Zahlenformat, das er benötigt und speichert ihn in der Variablen `wert` ab.
- Anschließend wird die Verbindung wieder abgebaut.

Modellieren und zeichnen Sie gemäß der obenstehenden Beschreibung für einen Temperatursensor unter der Verwendung der Klassen `ServerSocket` und `Socket` (Material 2) ein vollständiges UML-Sequenzdiagramm (Client- und Serverseite) in Material 3.

(15 BE)

2 Für die Bewässerung wird eine Pumpe in Kombination mit einem Durchflusssensor verwendet. Die Pumpe wird von einem Gleichstrommotor angetrieben, der ein- (1) und ausgeschaltet (0) werden kann. Der Durchflusssensor gibt 8 Pulse pro Liter Flüssigkeit, die ihn durchfließt, ab. Die steigende Flanke des Pulses gibt jeweils an, dass ein weiterer achtel Liter den Sensor passiert hat. Die Dauer des High-Pegels hängt dabei von der Durchflussgeschwindigkeit ab und ist somit unbestimmt.
Die Pumpe wird über das Unterprogramm `bewaesserung` gesteuert. Zu Beginn des Unterprogramms befindet sich die zu pumpende Wassermenge in Litern in einem festgelegten Register. Das Unterprogramm sorgt dafür, dass der Motor der Pumpe so lange läuft, bis die geforderte Menge Wasser den Durchflusssensor passiert hat.
Die Bedeutung der Signale und die Belegung des Mikrocontrollerports entnehmen Sie der folgenden Tabelle:

Port des Mikrocontrollers							
Pin 7	Pin 6	Pin 5	Pin 4	Pin 3	Pin 2	Pin 1	Pin 0
-	-	-	-	-	-	Durchflusssensor	Motor

2.1 Wählen Sie das Register bei dem von Ihnen im Unterricht verwendeten Mikrocontroller aus, in dem die zu pumpende Wassermenge gespeichert ist.

(2 BE)

2.2 Entwickeln und zeichnen Sie ein Struktogramm für das Unterprogramm `bewaesserung`.

(9 BE)

2.3 Überführen und implementieren Sie das Unterprogramm `bewaesserung` sowie die Initialisierung der verwendeten Ports in den von Ihnen im Unterricht verwendeten Assembler.

(9 BE)

3 Die Temperatur soll auch im Gewächshaus mittels 7-Segment-Anzeigen angezeigt werden. Die Messwerte des Temperatursensors (Material 1) werden von einer Auswerteschaltung in einen BCD-Code mit zwei Dezimalstellen übersetzt. Um damit die 7-Segment-Anzeigen anzusteuern, wird ein BCD-zu-7-Segment-Decoder benötigt (Material 4).

3.1 Entwickeln Sie die Tabelle für die Umwandlung des BCD-Codes in einen 7-Segment-Code in der Funktionstabelle in Material 5.

(7 BE)

3.2 Vereinfachen Sie die Schaltfunktionen für die Segmente a, b und c in den KV-Diagrammen in Material 6.

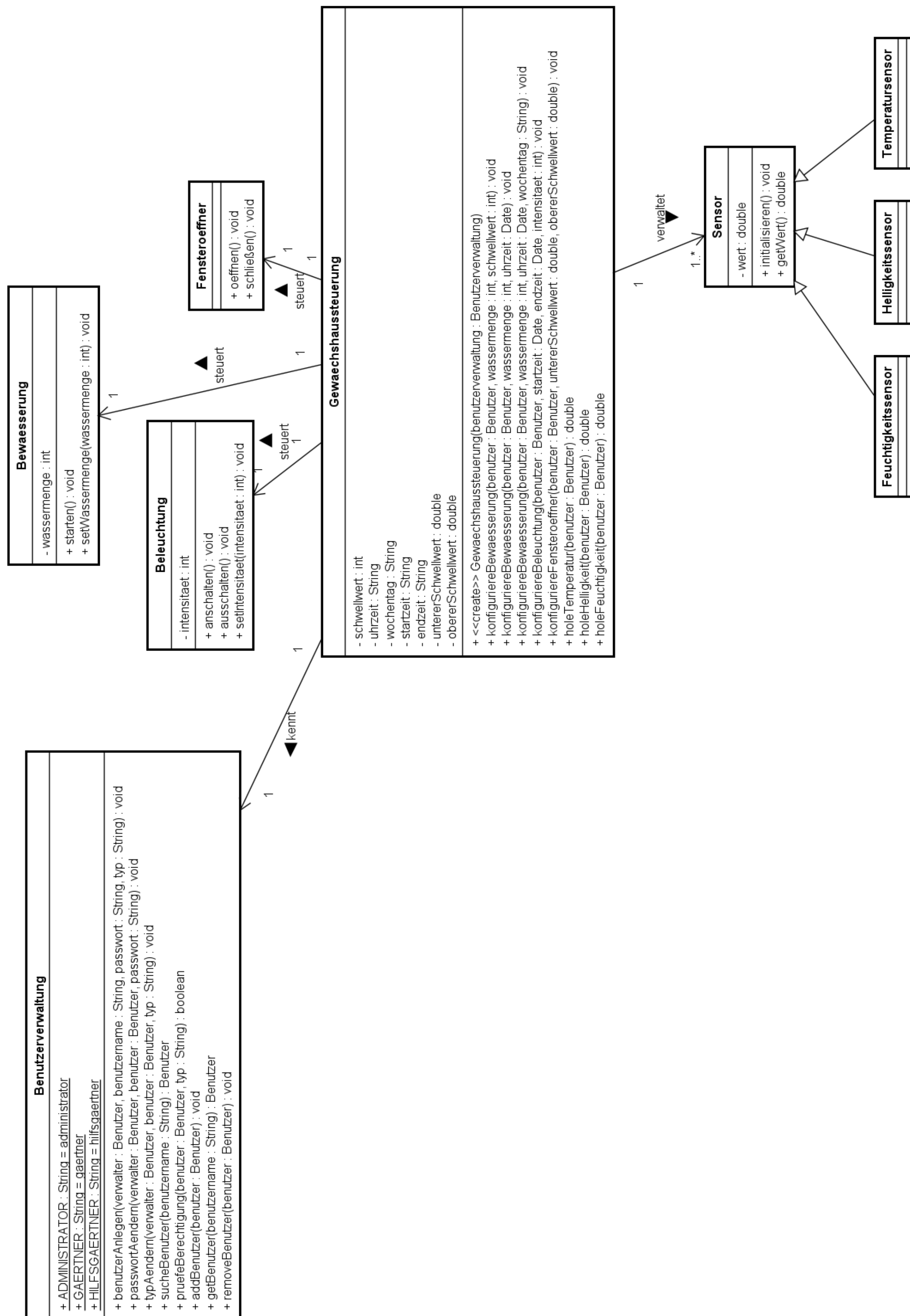
(12 BE)

3.3 Überführen Sie die Schaltfunktionen für die Segmente a, b und c in NAND-Form und zeichnen Sie diese.

(9 BE)

Material 1

UML-Klassendiagramm



Material 2

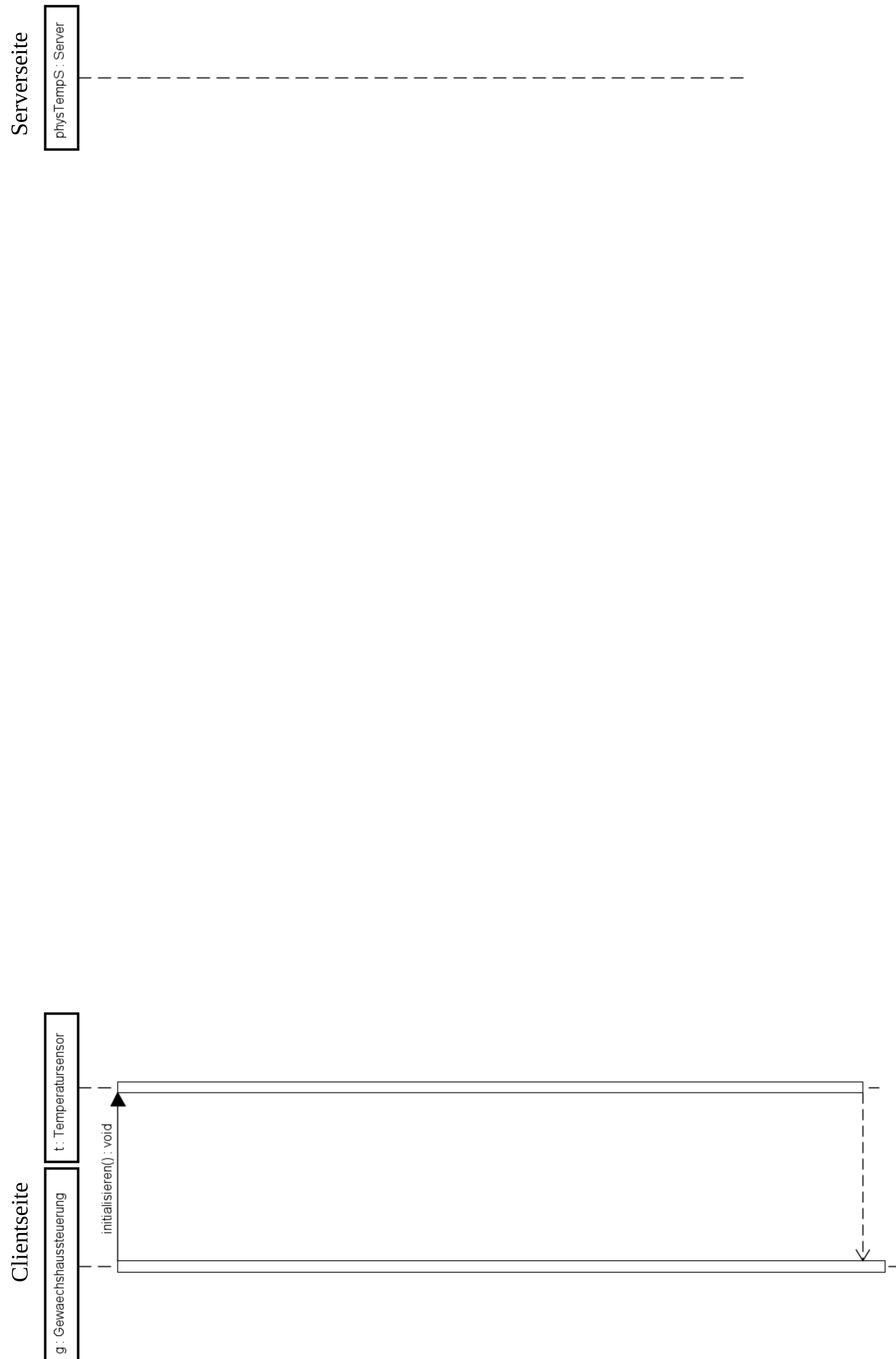
ServerSocket und Socket

ServerSocket
- localPort : int
+ <<create>> ServerSocket(localPort : int) : ServerSocket + accept() : Socket + close() : void

Socket
- remoteHostIP : String - remotePort : int
+ <<create>> Socket(remoteHostIP : String, remotePort : int) : Socket + connect() : boolean + dataAvailable() : int + read() : int + read(b : byte[], len : int) : int + readLine() : String + write(value : int) : void + write(b : byte[], len : int) : void + write(s : String) : void + close() : void

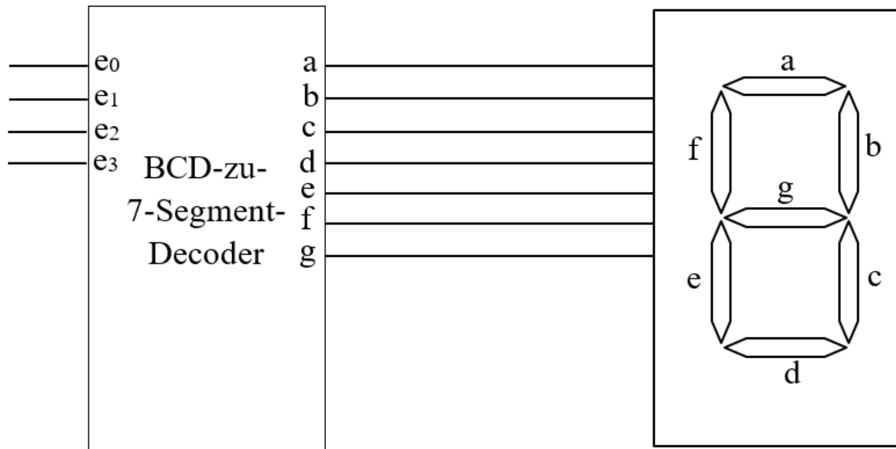
Material 3

UML-Sequenzdiagramm für Client-Server-Kommunikation



Material 4

BCD-zu-7-Segment-Decoder



Material 5

Funktionstabelle

BCD-Code				7-Segment-Anzeige						
e_3	e_2	e_1	e_0	a	b	c	d	e	f	g

Material 6

KV-Diagramme

